I. INTRODUCTION A « .Net »

1. L'environnement ou Framework Microsoft DotNet

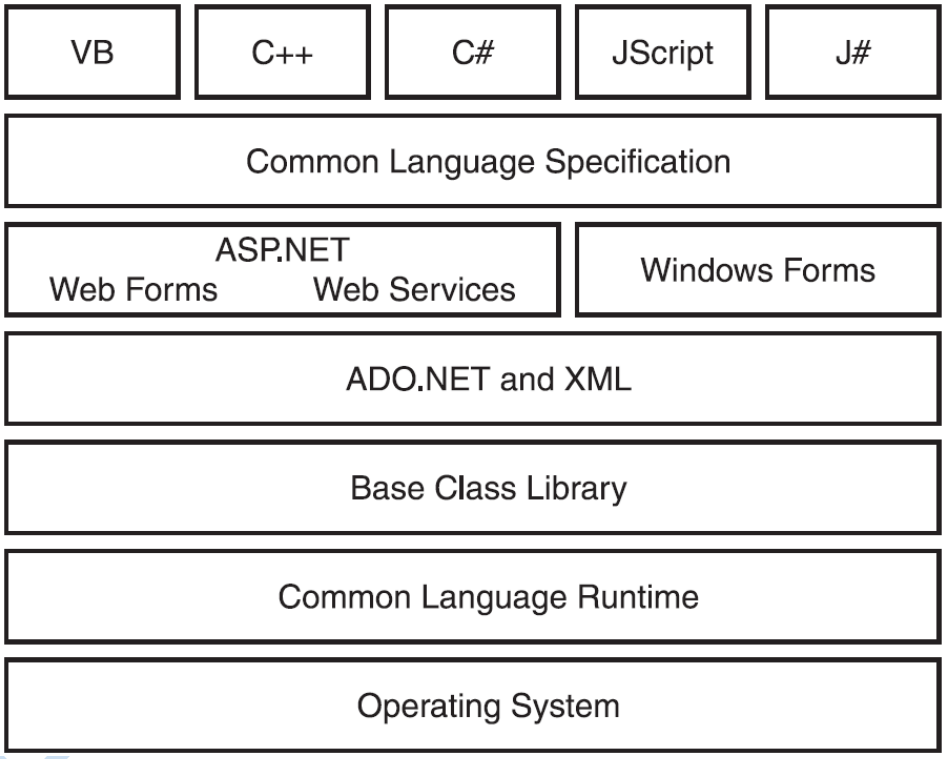
1. Définition : Framework

Définition
Un Framework est un **logiciel** composé d'ensemble d'outils ou de programmes qui fournissent au programmeur les fonctionnalités nécessaire pour réaliser des applications informatiques.

2. Le Framework .Net

Le Framework .Net (DotNet) est créé par Microsoft en 2002. Il contient un grand nombre d'outils, de technologies et de langages de programmation, qui servent à créer des applications destinées aux systèmes DOS, Windows, Internet ou encore WindosPhone.

Structure simplifiée des couches qui constituent le Framework .Net :



1 Plusieurs langages de programmation supportés, dans le but d'attirer le maximum de développeurs.

2 Pour aller plus loin avec la fonctionnalité **1**, le DotNet permet au programmeur de réaliser les parties de son projet avec plusieurs langages.
Par ex. : dans une seule application web, on peut écrire une page en **C#**, et une autre en ***VB.Net***, …
Bien sûr, ces langages ont des syntaxes et des structures différentes, il faut donc les traduire vers un langage unifié pour qu'ils soient interprétés de la même manière.
C'est le rôle de l'outil Common Language Specification **CLS**.

3 La 3ieme couche contient deux *bibliothèques de classes* :

La notion de classe sera détaillée dans le chapitre 3

Windows Forms : utilisée pour le développement d'applications Windows.
ASP.Net : utilisée pour le développement d'applications Web, et le développement et la consommation de Services Web.

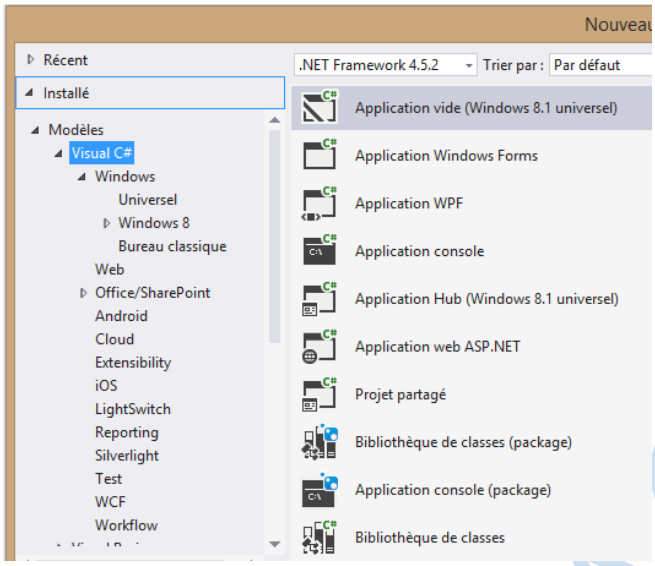
4 La couche suivante fournit au programmeur les bibliothèques nécessaires pour interagir avec un SGBD (***ADO.Net***), et pour lire et générer des fichiers XML.

5 La Base Class Library ***BCL*** contient toutes les classes utilisées par tous les langages.

6 La couche Common Language Runtime ***CLR*** assure la compilation, l'interprétation, l'exécution et la gestion des exceptions du code écrit par le programmeur. Car en fait, le système d'exploitation (Windows) ne sait exécuter que des instructions écrites en langage binaire, ce qui n'est pas le cas pour les instructions écrites par le programmeur, qui lui ne peut pas exprimer ses instructions en binaire.
Ce qu'il faut donc c'est un traducteur entre les deux, c'est le rôle du ***CLR***.

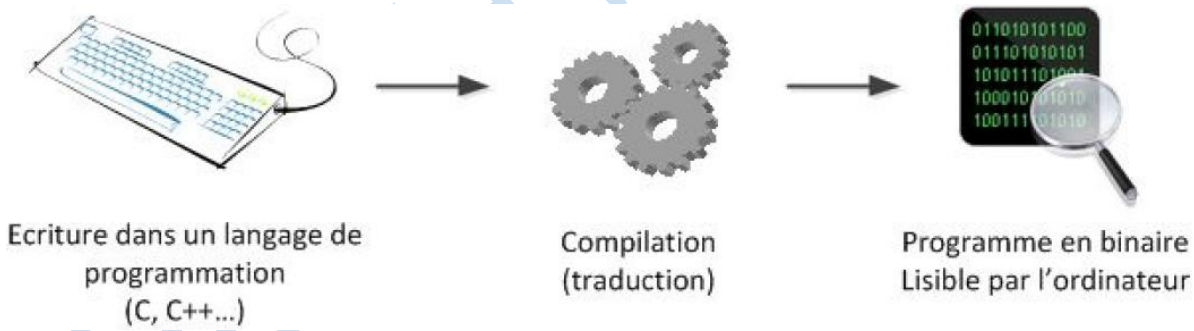
2. Le langage C#

- Créé en même temps et spécifiquement pour le Framework DotNet, c'est le langage phare de ce Framework.
- Utilisé dans tous genre de projets (MSDOS, Windows, Windows Phone, applications Web, application embarquées, applications Android, …)

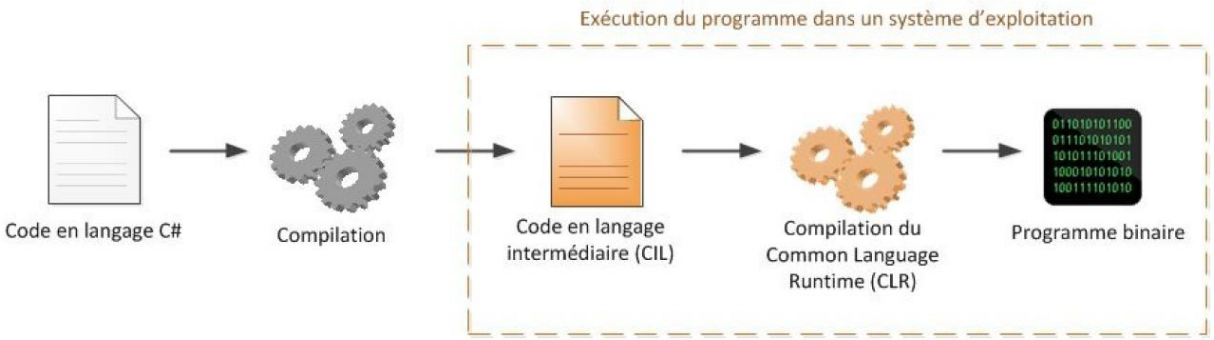


- Le langage C# est un langage « intermédiaire »

Rappel : Langage compilé (C, C++, VB, …)



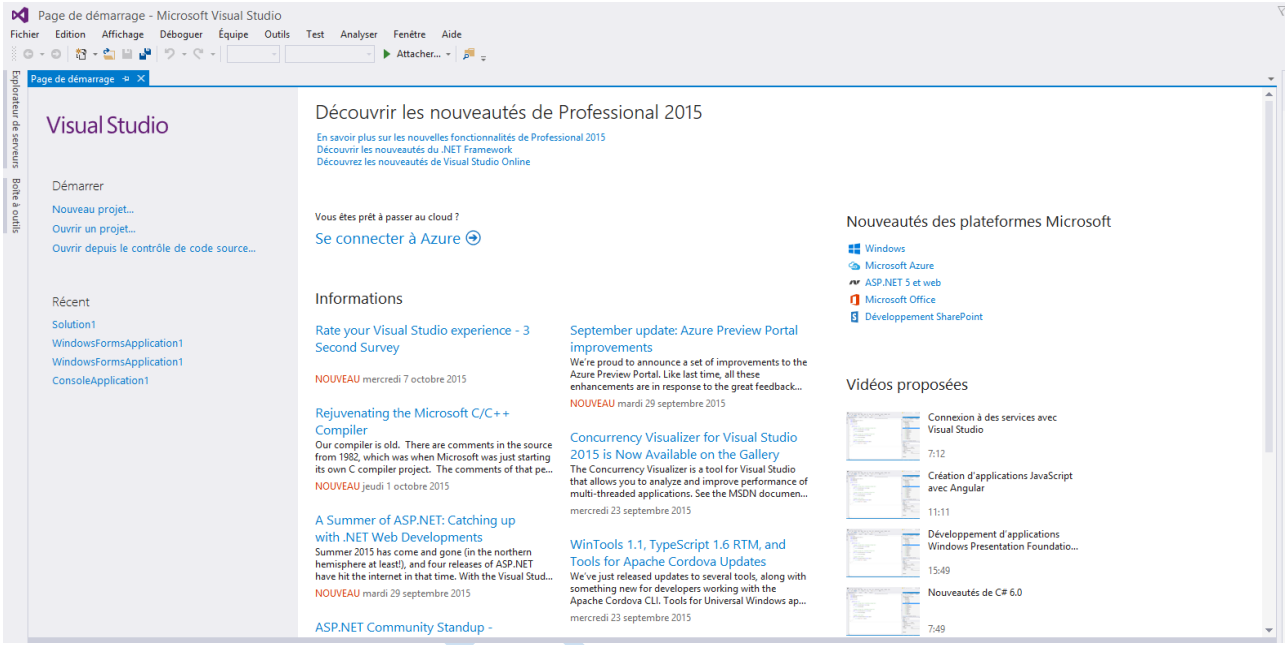
Langage intermédiaire (C#, VB.Net, …)



3. Microsoft Visual studio

Définition : IDE (Integrated Development Environment)
Environnement qui permet au programmeur de : • écrire, tester et corriger ses programmes, • créer, lier, déployer des projets • exemples : Visual Studio .Net, Eclipse, …

Pour utiliser les fonctionnalités offertes par le Framework .Net à travers un langage .Net, un programmeur a besoin de l'IDE Visual Studio .Net.



Visual Studio .Net est équipé de la part du ***CLR*** de deux fonctionnalités très utiles aux programmeurs :

L'auto-complétion : sert à assister le programmeur dans le choix des mots-clefs du langage utilisé.

```
class Program
{
    0 références
    static void Main(string[] args)
    {
        in
    }
}
```

indexer
IndexOutOfRangeException
InsufficientExecutionStackException
InsufficientMemoryException
int
Int16
Int32
Int64
interface
Mot clé int

Le compilateur CLR ***JIT*** (Just In Time) : analyse le code écrit et signale instantanément les erreurs de syntaxe.

```
class Program
{
    0 références
    static void Main(string[] args)
    {
        doubl x;
    }
}
```

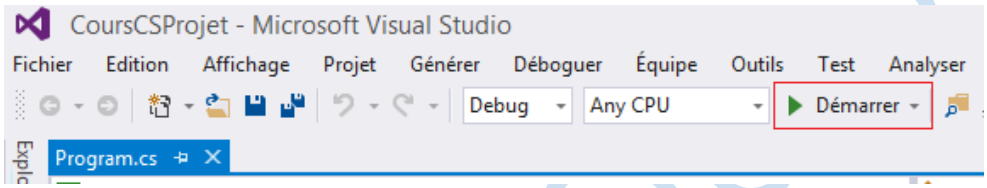
Le nom de type ou d'espace de noms 'doubl' est introuvable

Afficher les corrections éventuelles (Ctrl+.)

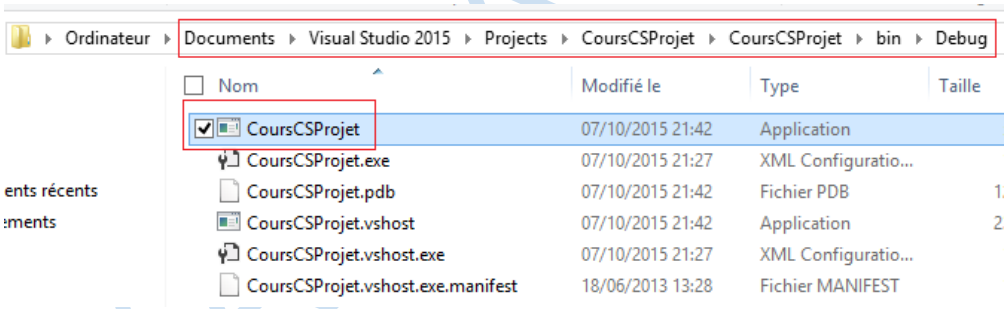
4. Créer un premier projet

Notions préliminaires :

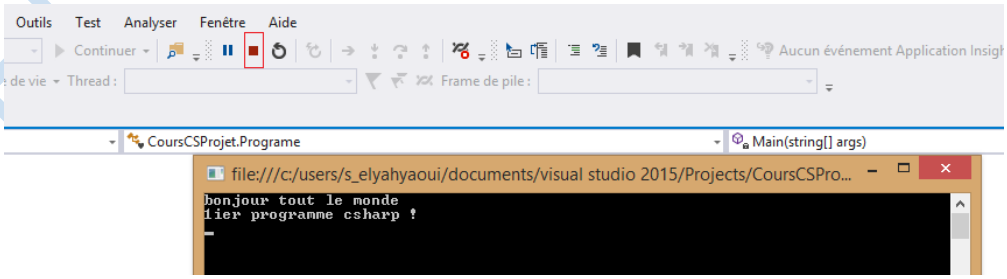
- Une application informatique est réalisée par la « collaboration » de plusieurs fichiers (contenant le code C#, VB, …), ces fichiers peuvent être réunis dans un seul projet.
- Pour être exécuté, un projet doit contenir au moins un fichier, et ce fichier doit obligatoirement contenir une classe, qui doit obligatoirement une fonction nommée ***main***. C'est la fonction principale du projet, elle constitue le point de départ lors de l'exécution du projet.
- Les projets traités pour l'instant sont les projets de type : « ***Application console*** », s'exécutant dans la console de ligne de commandes MSDOS.
- Pour **générer et lancer** l'exécutable du projet, on peut soit :
  - Utiliser la commande « **Démarrer** »



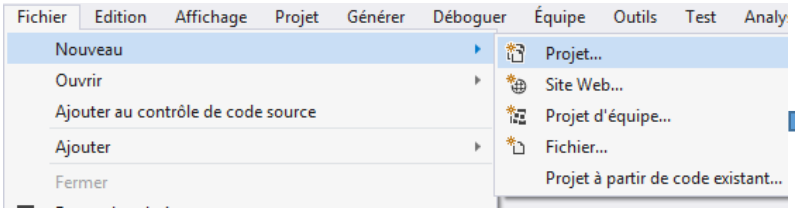
  - Soit la commande « **Générer la solution** » dans le menu « **Générer** », puis lancer manuellement l'exécutable créé dans le dossier « **debug** » du projet

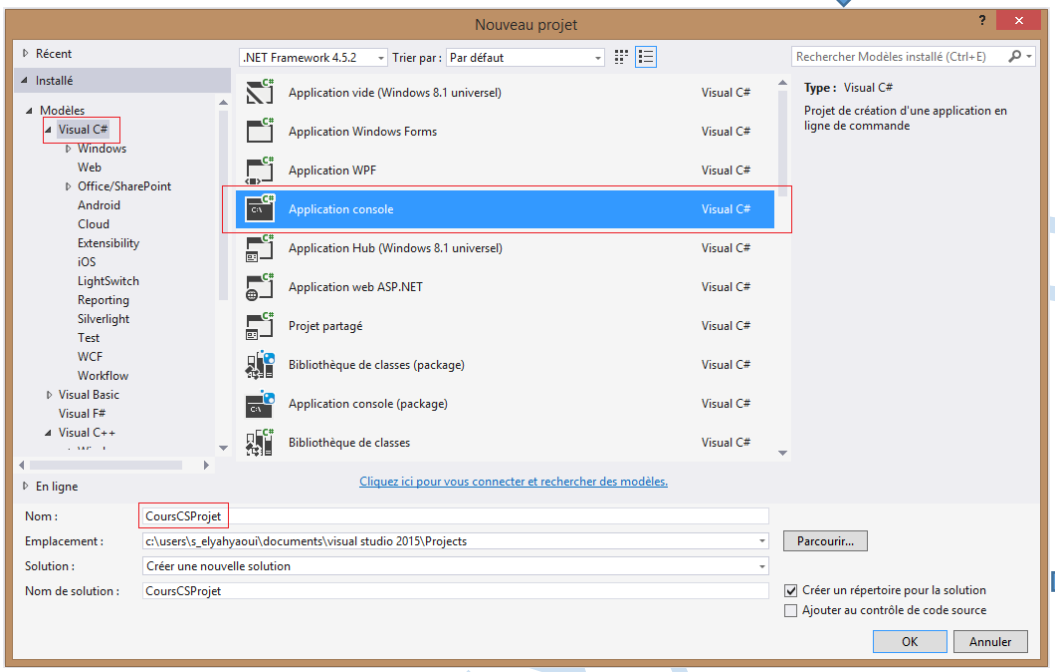


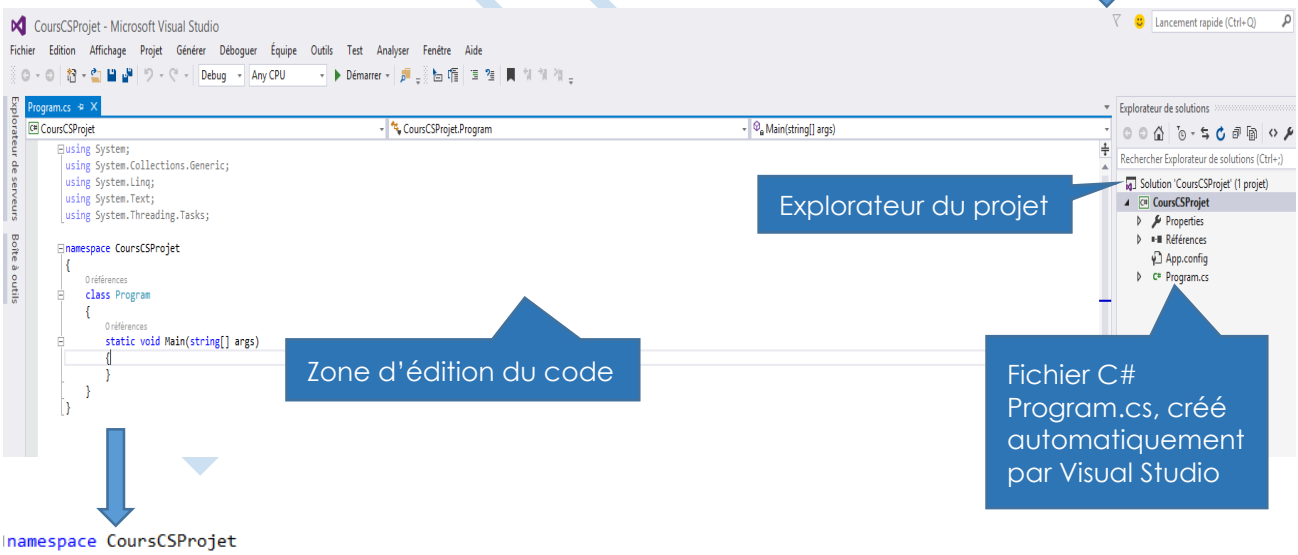
  - L'arrêt de l'exécution se fait fermeture de la fenêtre de console ou par la commande « Arrêter »



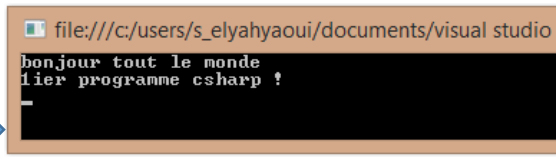
  - Il est **impossible** de modifier le code pendant l'exécution du programme.








```
namespace CoursCSProjet
{
    0 références
    class Programe
    {
        0 références
        static void Main(string[] args)
        {
            Console.WriteLine("bonjour tout le monde");
            Console.WriteLine("1ier programme csharp !");
            Console.ReadKey();
        }
    }
}
```



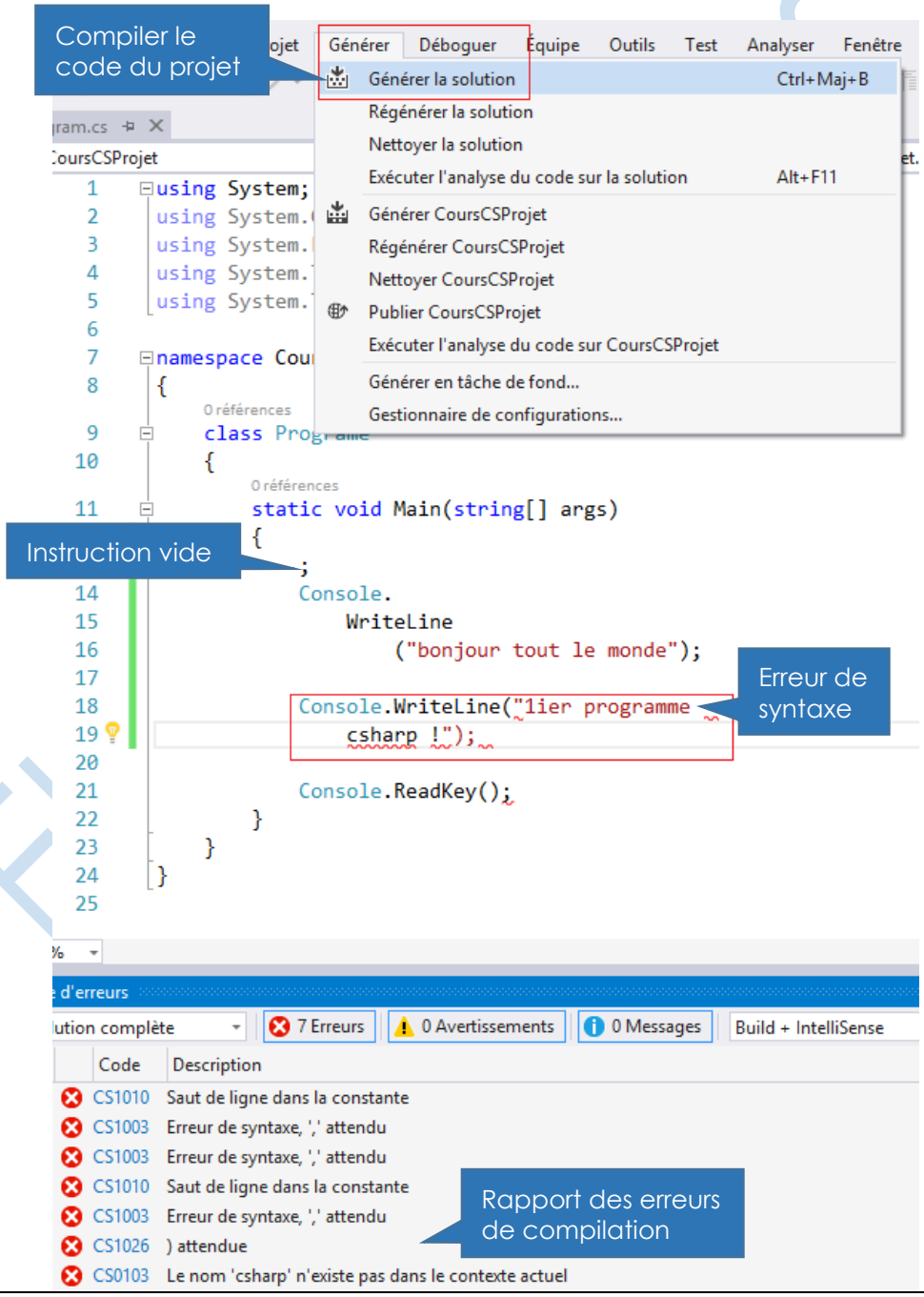
II. NOTIONS DE PROGRAMMATION STRUCTUREE EN CSHARP

1. La syntaxe de C#

1. Instructions

Notions préliminaires :

- Le compilateur JIT du CLR souligne en rouge les erreurs de syntaxe.
- Une instruction **doit toujours se terminer par un point-virgule**.
- L'instruction qui ne contient que le point-virgule est appelée « l'instruction vide ». Elle ne spécifie aucun ordre à exécuter de la part de l'ordinateur.
- Une instruction peut être écrite sur plusieurs lignes, sauf si elle contient une chaine de caractères (" **…** "), la chaine de caractères ne doit pas être interrompue par un retour à la ligne.



2. Commentaires

Définition

Un commentaire est une partie du code qui n'est pas prise en compte lors de la compilation. Il sert généralement à ajouter des explications concernant certaines instructions. C# supporte 2 types de commentaires :

- Commentaire sur une ligne **//**
- Commentaire sur plusieurs lignes **/* … */**

```
static void Main(string[] args)
{   /*
        une insturction
        vide
    */
    ;

    // affichage 1iere ligne
    Console.WriteLine("bonjour tout le monde");
    // affichage 2ieme ligne
    Console.WriteLine("1ier programme csharp !");
    // attendre qu'on tape une touche du clavier
    Console.ReadKey();
}
```

3. Indentation du code

✔ Code indenté

```
namespace CoursCSProjet
{
    0 références
    class Programe
    {
        0 références
        static void Main(string[] args)
        {   /*
                une insturction
                vide
            */
            ;

            // affichage 1iere ligne
            Console.WriteLine("bonjour tout le monde");
            // affichage 2ieme ligne
            Console.WriteLine("1ier programme csharp !");
            // attendre qu'on tape une touche du clavier
            Console.ReadKey();
        }
    }
}
```

✖ Code non indenté

```
namespace CoursCSProjet
 0 références
{class Programe
 0 références
{static void Main(string[] args)
{/*
une insturction
vide
*/
;
// affichage 1iere ligne
Console.WriteLine("bonjour tout le monde");
// affichage 2ieme ligne
Console.WriteLine("1ier programme csharp !");
// attendre qu'on tape une touche du clavier
Console.ReadKey();
}}}
```

IMPORTANT

Les commentaires et l'indentation sont des pratiques qui améliorent sensiblement la lisibilité, (donc la maintenabilité) du code.

2. Les variables

Définition : Variable
L'ordinateur consacre un « espace mémoire » dans la RAM pour chaque programme qu'il exécute. Bien sûr, tout programme doit gérer des informations (chiffres, noms, date de naissances, adresses, …). Chaque information est stockée dans une « zone » de l'espace mémoire consacré au programme. En programmation, cette zone est appelée : « **une variable** ». Avant d'utiliser une variable dans un programme, on doit la « **déclarer** », pour qu'elle soit prise en compte par le compilateur.

1. Déclaration & Affectation

Syntaxe de base

Déclaration :
```
type_de_donnée  nom_var;
```
Ou bien :
```
type_de_donnée  nom_var1, nom_var2, ...;
```

Affectation : stocker une valeur à une variable.
```
nom_var = valeur;
```
Ou bien :
```
nom_var1 = nom_var2;
```
Affecter la valeur d'une variable à une autre variable :
nom_var1 reçoit la valeur de ***nom_var2***

Déclaration + affectation :
```
type_de_donnée  nom_var = valeur;
```

2. Types de variables

Type	Définition
byte	Entier non-signé entre 0 et 255 (1 Octet)
short	Entier entre -32768 et 32767
int	Entier entre -2147483648 et 2147483647
long	Entier entre -9223372036854775808 et 9223372036854775807
float	Réel en simple précision entre -3,402823e38 et 3,402823e38
double	Réel en double précision entre -1,79769313486232e308 et 1,79769313486232e308
decimal	Autre type réel, créé spécifiquement pour les calculs financiers. N'accepte pas certaines incertitudes de calcul, acceptées par les types **float** et **double**.
char	Caractère unique (alphanumérique). (**'a'**, **'A'**, **'9'**, **'@'**, …) Toujours placé entre simples cotes : ' ' Le caractère vide est représenté par **'\0'**
string	Chaine de caractères. Toujours placée entre doubles cotes : "abc123…" La chaine vide est représentée par : ""
bool	Booléen. N'accepte que l'une des valeurs logiques : **true** ou **false**.

3. Afficher des informations à l'écran

En mode console, l'affichage de toute information se fait par les fonction ***Write( )*** et ***WriteLine( )*** de la classe **Console**.

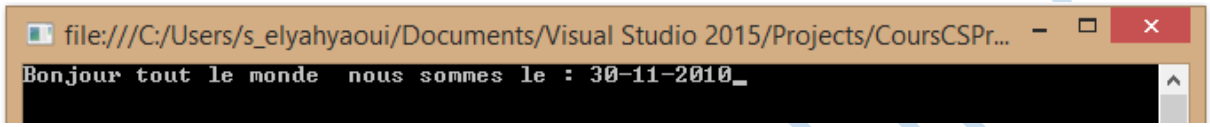
1. Fonction Write

Définition

Afficher une information dans la console.
Syntaxe de base :
Console.Write(…)

Exemple

Affichage du message



En utilisant les variables suivantes :

Variable	Valeur
s1	bonjour
s2	tout le monde
s3	nous sommes le :
a	30
b	11
c	2010

```
class Programe
{
    0 références
    static void Main(string[] args)
    {
        string s1 = "Bonjour ";
        string s2 = "tout le monde ", s3 = " nous sommes le : ";
        short a = 30, b = 11, c = 2010;
        Console.Write(s1);
        Console.Write(s2);
        Console.Write(s3);
        Console.Write(a);
        Console.Write("-");
        Console.Write(b);
        Console.Write("-");
        Console.Write(c);
        Console.ReadKey();
    }
}
```

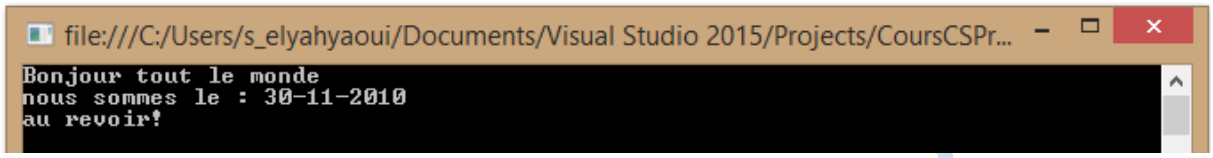
2. Fonction WriteLine

Définition

Afficher une information suivie d'un retour à la ligne.
Syntaxe de base :
Console.WriteLine(…)

Exemple

Affichage du message



En utilisant les mêmes variables de l'exemple précédent

```
string s1 = "Bonjour ", s2 = "tout le monde ", s3 = "nous sommes le : ";
short a = 30, b = 11, c = 2010;
Console.Write(s1); Console.WriteLine(s2); Console.Write(s3);
Console.Write(a); Console.Write("-"); Console.Write(b); Console.Write("-");
Console.WriteLine(c);
Console.Write("au revoir!");
Console.ReadKey();
```

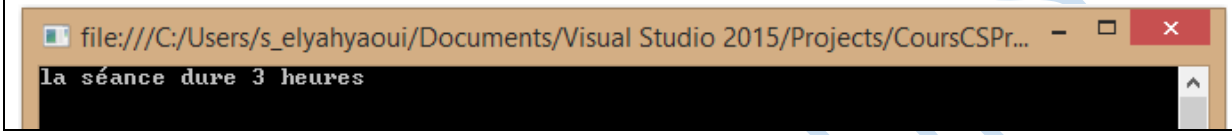
3. Opérateur de concaténation

Définition

La concaténation est l'action de « coller » une chaine de caractères à un chiffre, ou une autre chaine de caractères, ou une variable d'un autre type.
Cette opération est faite par l'opérateur **+**
Syntaxe de base :
valeur1 + valeur2 + …

Exemple

```
int duree = 3;
Console.Write("la séance dure " + duree + " heures");
Console.ReadKey();
```

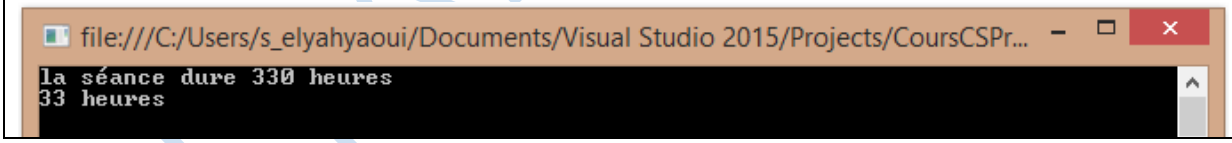


IMPORTANT

Utilisé entre deux chiffres, l'opérateur **+** agira comme opérateur d'addition et non de concaténation.

Exemple

```
int duree = 3, minutes = 30;
Console.WriteLine("la séance dure " + duree + minutes + " heures");
Console.WriteLine(duree + minutes + " heures");
Console.ReadKey();
```

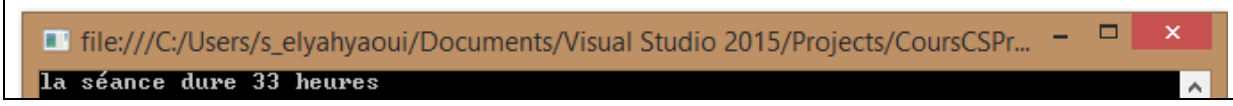


REMARQUE

L'utilisation des parenthèses ( ) annule l'effet de la concaténation.

Exemple

```
int duree = 3, minutes = 30;
Console.Write("la séance dure " + (duree + minutes) + " heures");
Console.ReadKey();
```

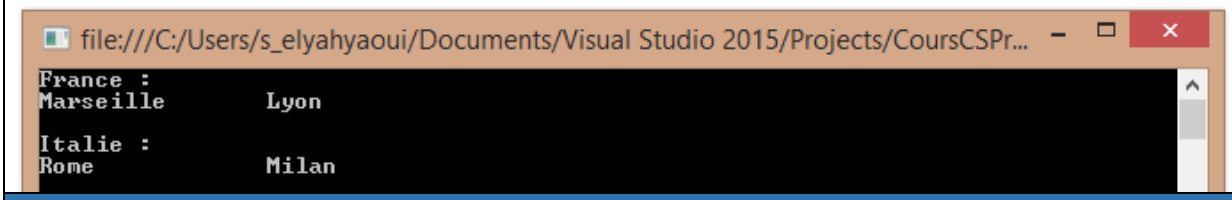


4. Caractères spéciaux

\t Tabulation, provoquer un nombre d'espaces dans l'affichage.
\n Retour à la ligne.

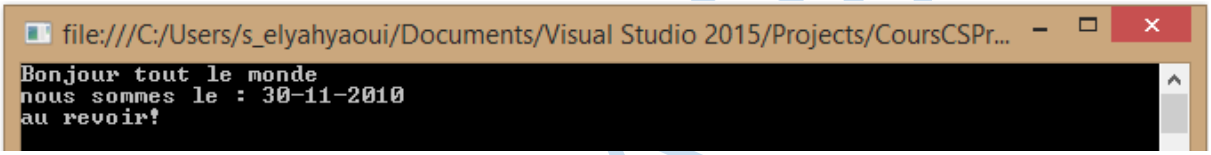
Exemple 1

```
Console.Write("France :\n");
Console.Write("Marseille\tLyon\n\nItalie :\nRome\t\tMilan");
Console.ReadKey();
```



Exemple 2

Afficher le message



En utilisant les mêmes variables de l'exemple précédent, en utilisant une seule fois la fonction ***Write()***.

```
string s1 = "Bonjour ";
string s2 = "tout le monde ", s3 = "nous sommes le : ";
short a = 30, b = 11, c = 2010;
Console.Write(s1 + s2 + "\n" + s3 + a + "-" + b + "-" + c + "\nau revoir!");
Console.ReadKey();
```

5. Autres fonctions de la classe Console

Beep() Emet un signal sonore.
Clear() Efface les messages affichés sur la console.

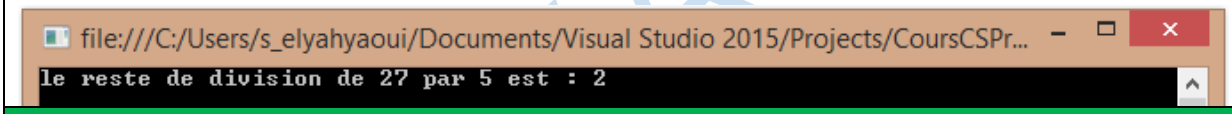
4. Les opérateurs numériques et logiques

1. Les opérateurs numériques

Définitions - 1	
=	*opérateur d'affectation*
%	*modulo : reste de division euclidienne*
==	*égale : test d'égalité entre deux chiffres*
!=	*différent de : test d'inégalité entre deux chiffres*
<	*inférieur*
>	*supérieur*
<=	*inférieur ou égale*
>=	*supérieur ou égale*
+	*addition*
*	*multiplication*
-	*soustraction*
/	*division*

Exemple 1

```
byte x1 = 27, x2 = 5;
Console.Write("le reste de division de 27 par 5 est : " + x1 % x2);
Console.ReadKey();
```



Définitions - 2

+= (incrémentation)	`x += a;`	`x = x + a;`	`int x = 1;` `x += 5; // la valeur de x est 6`
-= (décrémentation)	`x -= a;`	`x = x - a;`	`int x = 9;` `x -= 5; // la valeur de x est 4`

++ (incrémentation par **1**)	`x++;`
	ou bien
	`++x;`
		`x = x + 1;`	`int x = 9;`
			`x++; // la valeur de x est 10`
			`++x; // la valeur de x est 11`

-- (décrémentation par **1**)	`x--;`
	ou bien
	`--x;`
		`x = x - 1;`	`int x = 9;`
			`x--; // la valeur de x est donc 8`
			`--x; // la valeur de x est donc 7`

Exercice

Déterminer les valeurs de la variable **x** après chaque instruction :

`int x = 0;`	**0**
`x += 10;`	…
`++x;`	…
`x--;`	…
`x += 2;`	…
`x++;`	…

Définitions - 3

*=	x *= a;	x = x * a;	int x = 10; x *= 5; // la valeur de x est 50
/=	x /= a;	x = x / a;	int x = 12; x /= 4; // la valeur de x est 3
%=	x %= a;	x = x % a;	int x = 27; x %= 5; // la valeur de x est 2

Exercice

Déterminer les valeurs de la variable **x** après chaque instruction :

int x = 0;	**0**
x *= 10;	…
x /= 5;	…
x %= 2;	…
x += 10;	…
x *= 1.25;	…

2. Pré / Post incrémentation - décrémentation

IMPORTANT

1- Les opérateurs **++** et **--** peuvent être utilisés dans des opérations d'affectation et de calcul.
Exemple :

```
int x = 10;
int n = ++x;
```

Le résultat de l'opération dépend de l'emplacement de l'opérateur **++**
C.à.d. **++x** (dans ce cas on parle de ***Pré-incrémentation***) ou bien **x++** (dans ce cas on parle de ***Post-incrémentation***)

Exemple :

<table>
<tr><th colspan="2">Décrémentation (opérateur --)</th></tr>
<tr><th>Pré</th><th>Post</th></tr>
<tr><th colspan="2">Cas d'utilisation : Opération d'affectation</th></tr>
<tr>
<td>

```
int x = 10, n = 0;
n = --x;
```

x	n
10	0
9	9

L'opération d'affectation de **n** (`n = …`), utilise ici la nouvelle valeur de **x**, c.à.d. la valeur de ***x après*** la décrémentation
</td>
<td>

```
int x = 10, n = 0;
n = x--;
```

x	n
10	0
9	10

L'opération d'affectation de **n**, utilise ici **l'ancienne** valeur de **x**, c.à.d. la valeur de ***x avant*** la décrémentation
</td>
</tr>
<tr><th colspan="2">Cas d'utilisation : Opérations arithmétiques</th></tr>
<tr>
<td>

```
int x = 10, n = 0;
n = 2 * --x;
```

x	n
10	0
9	18

L'opérateur de multiplication (… `*` …), utilise ici la nouvelle valeur de **x**, c.à.d. la valeur de ***x après*** la décrémentation
</td>
<td>

```
int x = 10, n = 0;
n = 2 * x--;
```

x	n
10	0
9	20

L'opérateur de multiplication (… `*` …), utilise ici **l'ancienne** valeur de **x**, c.à.d. la valeur de ***x avant*** la décrémentation
</td>
</tr>
<tr><th colspan="2">Cas d'utilisation : Affichage à l'écran</th></tr>
<tr>
<td>

```
int x = 0;
Console.Write(--x);
```

Valeur de **x** : **-1**
Valeur affichée à l'écran : **-1**

La fonction d'affichage utilise ici la nouvelle valeur de **x**, c.à.d. la valeur de ***x après*** la décrémentation.
</td>
<td>

```
int x = 0;
Console.Write(x--);
```

Valeur de **x** : **-1**
Valeur affichée à l'écran : **0**

L'opération d'affichage, utilise ici **l'ancienne** valeur de **x**, c.à.d. la valeur de ***x avant*** la décrémentation.
</td>
</tr>
</table>

Résumé : (*Les cas étudiés sont valables pour l'opérateur ++ également*)
Les opérateurs ++ et -- ne sont prioritaires par rapport aux autres opérations que quand il s'agit de pré-incrémentation ou de pré-décrémentation.

2- Les opérateurs **+=** , **-=** , ***=** , **/=** et **%=** sont **toujours prioritaires** par rapport à l'opérateur d'affectation **=**

Exercice 1

Déterminer les valeurs des variables à chaque instruction du programme :

```
{
  int i = 1;

  int n = i++;

  i = 10;  n = ++i;

  i = 20;  j = 5;  n = i++ * ++j;

  i = 15;  n = i += 3;

  i = 3; j = 5;  n = i *= --j;
}
```

i	j	n
1		
2		…
…		…
…	…	…
…	…	…
…	…	…

Exercice 2

```
{
  byte i = 4, j = 1, k = 31, n;
  n = … ;
}
```

En utilisant les **3** variables **i**, **j** et **k**, et l'opérateur **++** ou **--**, compléter la 2ième instruction pour que la variable n prenne la valeur **150**.

3. Les opérateurs logiques

Définition 1 : Expression logique

Une expression logique est une expression qui n'a comme valeur possible que l'une des valeurs **Vrai** ou **Faux**.

Exemple :

```
byte x = 5;
byte y = 2;
bool z = (x == y);
```

Expression logique

un opérateur logique est un opérateur qui agit sur des variables et des expressions logiques.

Définition 2 : Opérateurs logiques

le ***NON***
!

A	! A : *non A*
VRAI	FAUX
FAUX	VRAI

le ***OU***
|| ou bien |

A	B	A || B : *A ou B*
VRAI	VRAI	VRAI
VRAI	FAUX	VRAI
FAUX	VRAI	VRAI
FAUX	FAUX	FAUX

le ***ET***
&& ou bien **&**

A	B	A && B : *A et B*
VRAI	VRAI	VRAI
VRAI	FAUX	FAUX
FAUX	VRAI	FAUX
FAUX	FAUX	FAUX

le ***OU EXCLUSIF***
^

A	B	A ^ B : *A ou bien B*
VRAI	VRAI	FAUX
VRAI	FAUX	VRAI
FAUX	VRAI	VRAI
FAUX	FAUX	FAUX